Introduction to Symbolic Verification Methods

David Basin

Institute of Information Security
ETH Zurich

Road map

☞ **Motivation**

- Basic notions
- Problems
- Symbolic models

Security protocols

- Omnipresent
  - **Authentication:** smart-card $\leftrightarrow$ ATM, single sign-on, ...
  - **Secure communication:** SSL/TLS, SSH, IPsec, ...
  - **Special purpose**: e-auctions, e-voting, ...
- Use **cryptographic** primitives to achieve security objectives
- Nontrivial to get right

  "Security protocols are three-line programs that people still manage to get wrong."

  Roger Needham

An example: naive use of primitives

- Consider following use of **Sign and Encrypt**

  $Alice \rightarrow Bob$: $\{\{\text{“I love you”}\}_{K^{-1}_{Alice}}\}_{K_{Bob}}$

  Alice signs and encrypts for Bob's eyes.

- Bob decrypts, re-encrypts, and forwards message to Charlie, who buys Alice flowers.

  $Bob \rightarrow Charlie$: $\{\{\text{“I love you”}\}_{K^{-1}_{Alice}}\}_{K_{Charlie}}$

- Protocol weakness has **nothing** to do with crypto building blocks
  - A protocol does more than just encrypt or sign.
  - It binds messages to principals, purposes, time, etc.

Goals for two classes

- To understand the kinds of problems that arise.
- To be precise about concepts and guarantees, where possible.
- To explain ideas behind different symbolic methods
  - **Methods/tools:** Paulson's inductive method, Scyther
- (Part II) To examine realistic protocols and problems that arise when humans are involved.
  - **Method/Tool:** Tamarin

Road map

- Motivation

☞ **Basic notions**

- Problems

- Symbolic models

Security protocols

- A **protocol** consists of rules describing how messages are exchanged between principals.

$$
\begin{array}{lll}
1. & A \to B: & \{A, N_A\}_{K_B} \\
2. & B \to A: & \{N_A, N_B\}_{K_A} \\
3. & A \to B: & \{N_B\}_{K_B}
\end{array}
$$

- A **security** (or **cryptographic**) protocol uses cryptographic mechanisms to achieve security objectives.

- In practice, descriptions combine prose, data types, diagrams, ad hoc notations, and message sequences as above.

Message constructors (sample)

Names: A, B or *Alice*, *Bob*, ...

Asymmetric keys: A's public key K_A and private key K_A^{-1}.

Symmetric keys: K_{AB} shared by A and B.

Encryption: asymmetric $\{M\}_{K_A}$ and symmetric $\{M\}_{K_{AB}}$.

Signing: $\{M\}_{K_A^{-1}}$.

Nonces: N_A. Fresh data items used for challenge/response.

Timestamps: T. Denote time, e.g., used for key expiration.

Message concatenation: M_1, M_2. (Or $M_1||M_2$)

Example: $\{A, T_A, K_{AB}\}_{K_B}$.

Communication

- Fundamental notion: communication between principals.

$$A \to B : \{A, T_A, K_{AB}\}_{K_B}$$

- A and B name **roles**.

  Can be instantiated by any principal playing the role.

- Communication is asynchronous.

  (Sometimes modeled as being synchronous.)

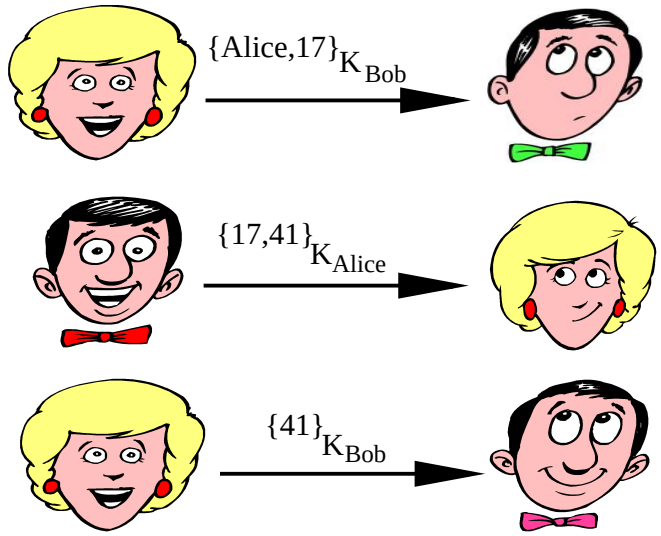
- Protocol specifies actions of principals in different protocol roles.

  It thereby also defines a set of event sequences (traces).

An authentication protocol (NSPK)

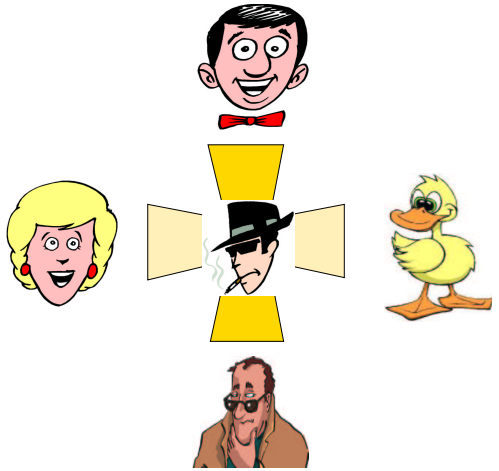
$$
\begin{array}{lll}
1. & A \to B: & \{A, N_A\}_{K_B} \\
2. & B \to A: & \{N_A, N_B\}_{K_A} \\
3. & A \to B: & \{N_B\}_{K_B}
\end{array}
$$

Here is an instance (a protocol run):



N.B. principals can be engaged in multiple runs (role automata).

Standard symbolic attacker model (Dolev-Yao)



- An active attacker who controls the network.
  - He can **intercept** and **read** all messages.
  - He can **decompose** messages into their parts.
    But cryptography is "perfect": decryption requires inverse keys.
  - He can **construct** and **send** new messages, any time.
  - He can even **compromise** some agents and learn their keys.

- A protocol should ensure that communication between **non-compromised** agents achieves objectives (next slide).

- Strong attacker $\Longrightarrow$ protocols work in many environments.

  Note: symbolic model idealizes cryptographic model based on bit-strings and probabilistic polynomial-time attackers.

Typical protocol objectives

Terminology not completely standard, but following are typical.

Entity authentication: One party verifies the identity of a second party and that this party has recently, actively participated in the protocol. ("I am here now.")

Secrecy (Confidentiality): Data available only to those authorized to obtain it. For keys, this is sometimes called **key authentication**.

Freshness: Data is new, i.e., not replayed from an older session.

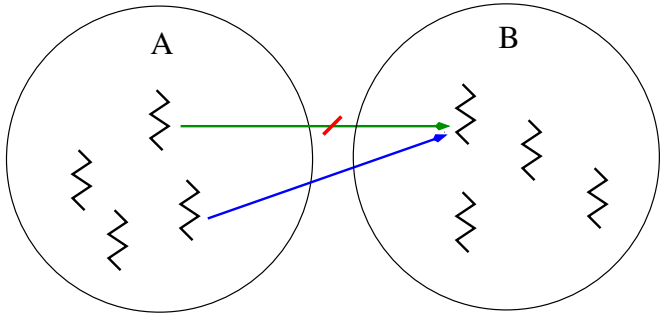
Key confirmation: One party is assured that a second party actually possess a given key.

Protocol objectives: entity authentication

- **Agreement** is a variant of authentication focusing on views.

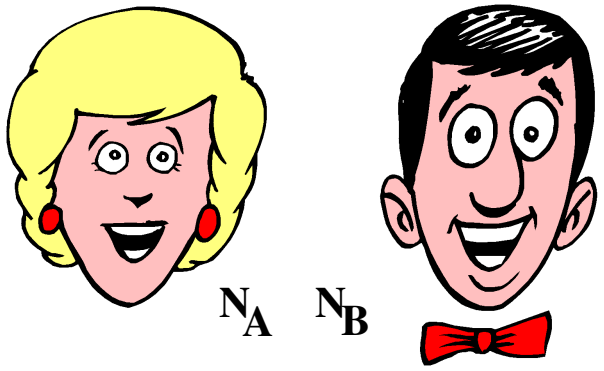
  A protocol guarantees that **an initiator A has non-injective agreement with a responder B on a set of data items ds** if, whenever A (acting as **initiator**) **completes** a run of the protocol, apparently with responder B, then B has been **running** the protocol, apparently with A, and B was acting as **responder** in his run, and the two agents **agreed** on the data values corresponding to all the variables in ds.

- **Injective agreement** when additionally each run of A corresponds to a **unique** run of B. Analogous notion of **matching histories** sometimes used.



Mechanisms used: nonces or timestamps with replay caches

Example: NSPK



1. $A \to B: \quad \{A, N_A\}_{K_B}$
2. $B \to A: \quad \{N_A, N_B\}_{K_A}$
3. $A \to B: \quad \{N_B\}_{K_B}$

Objective: Upon completion, A **injectively agrees** with B on both nonces, which are shared **secrets** between them. And vice versa.

Road map

- Motivation
- Basic notions
- ☞ **Problems**
- Symbolic models

Recall NSPK

1. $A \to B: \quad \{A, N_A\}_{K_B}$
2. $B \to A: \quad \{N_A, N_B\}_{K_A}$
3. $A \to B: \quad \{N_B\}_{K_B}$

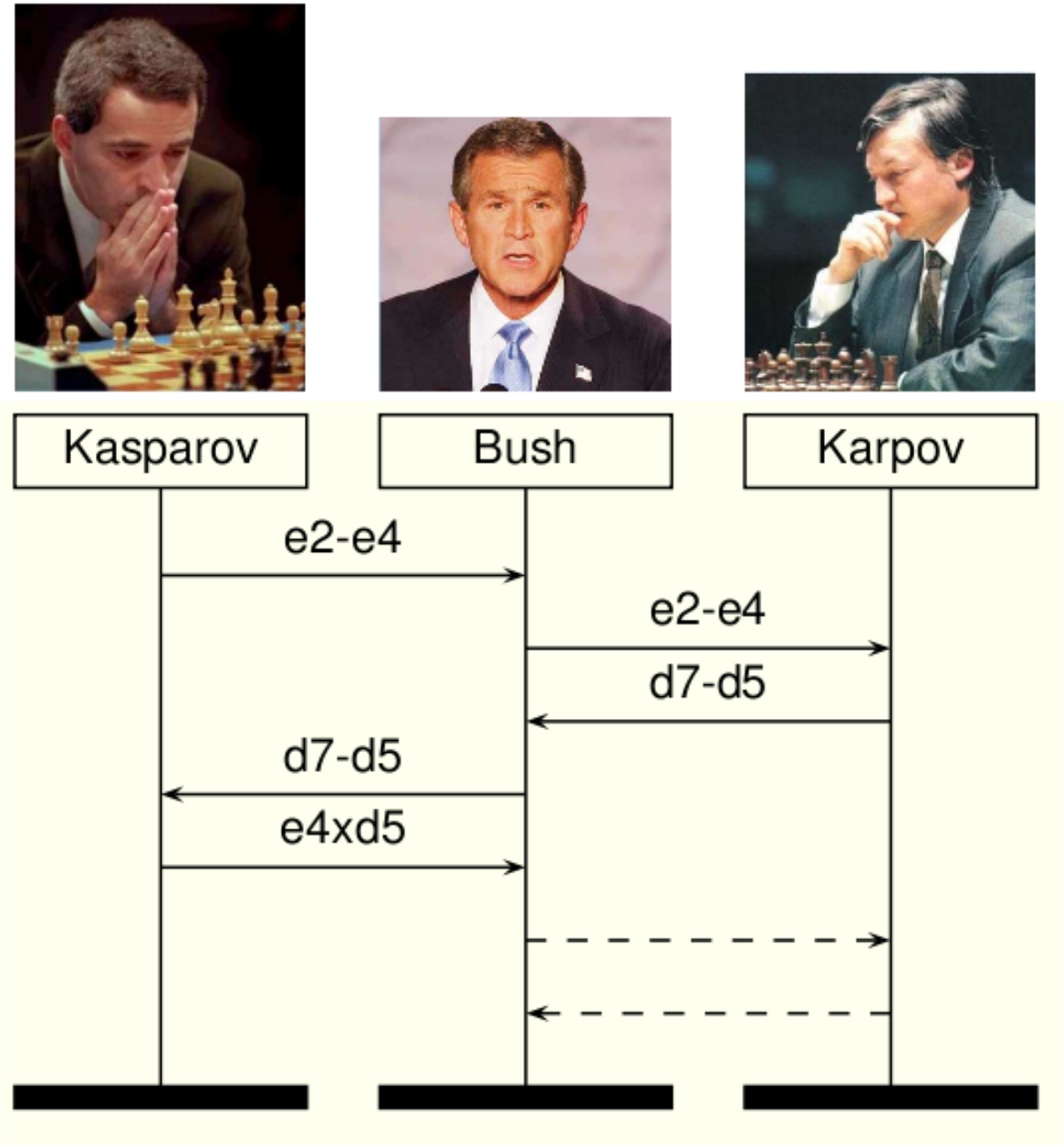
- Goal: mutual authentication (agreement).
- Recall principals can be involved in multiple runs.

  Goal should hold in all interleaved protocol runs.
- Correctness argument (informal).

  1. This is Alice and I have chosen a nonce N_{Alice}.
  2. Here is your Nonce N_{Alice}. Since I could read it, I must be Bob. I also have a challenge N_{Bob} for you.
  3. You sent me N_{Bob}. Since only Alice can read this and send it back, you must be Alice.

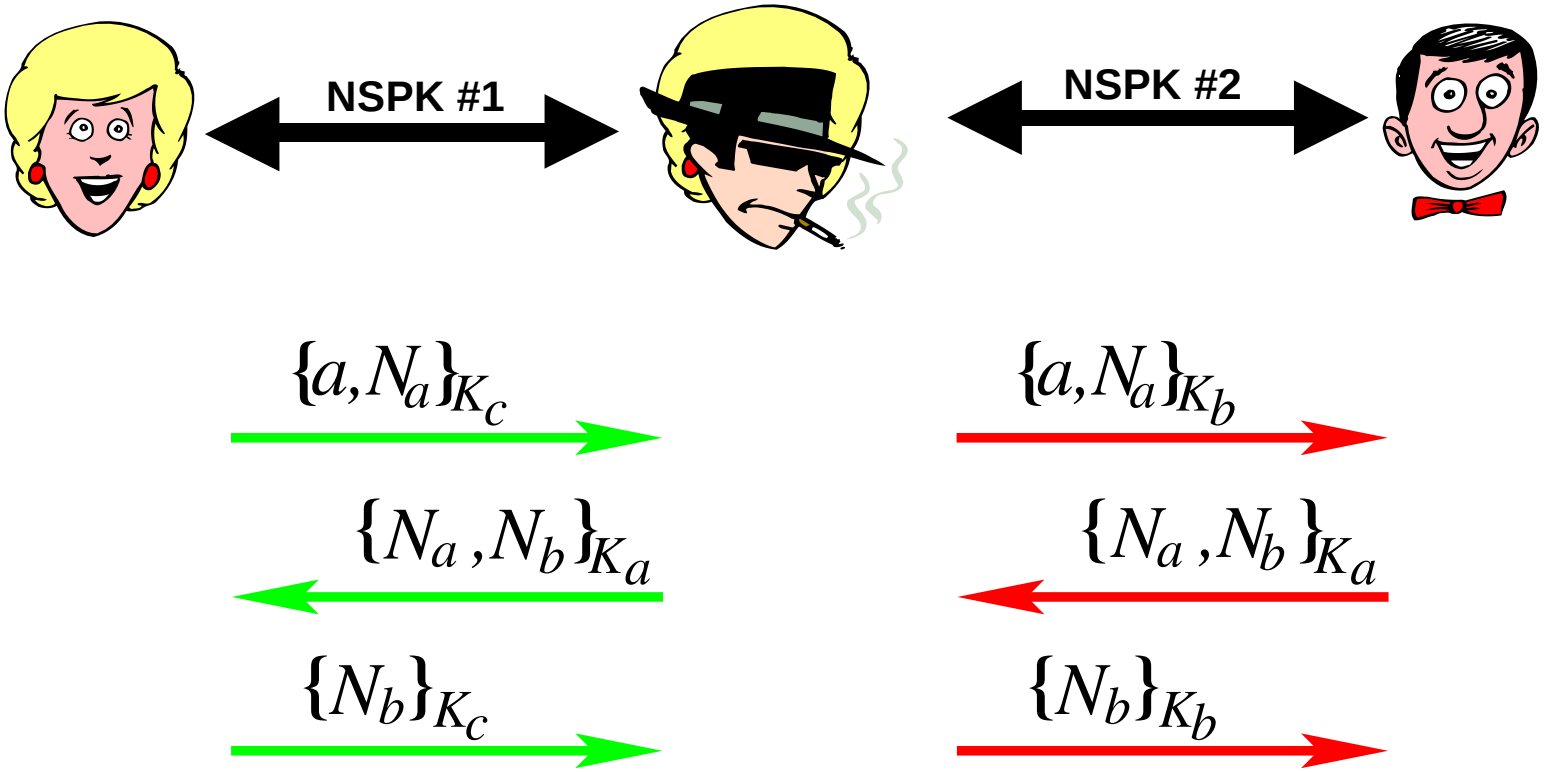
Protocol proposed in 1970s and used for decades.

Even Bush can beat a grandmaster



Attack on NSPK

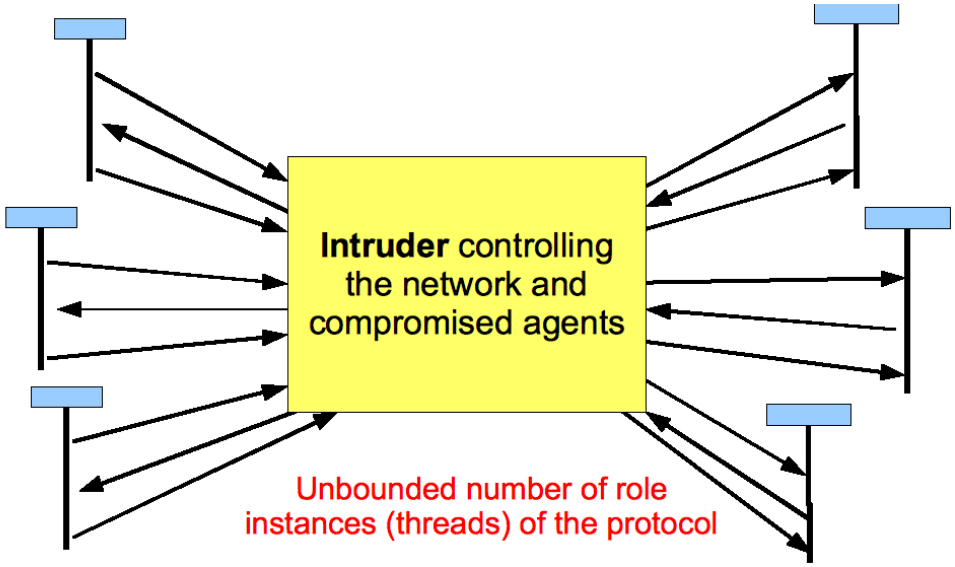
1. $A \rightarrow B: \quad \{A, N_A\}_{K_B}$
2. $B \rightarrow A: \quad \{N_A, N_B\}_{K_A}$
3. $A \rightarrow B: \quad \{N_B\}_{K_B}$



$b(ob)$ believes he is speaking with $a(lice)$!

How might you protect against this attack?

Why are such attacks so difficult to spot?



(It took 20 years to find attack.)

- Assumptions are unclear.

  Is the intruder an insider or an outsider?

- Complex underlying model despite the suggestion of simplicity.

- Humans poor at envisioning all possible interleaved computations.

- And real protocols are **much** more complex!
  - E.g., IPsec contains many messages, multiple subprotocols, etc.
  - Complexity reflects problems in design & standardization process.

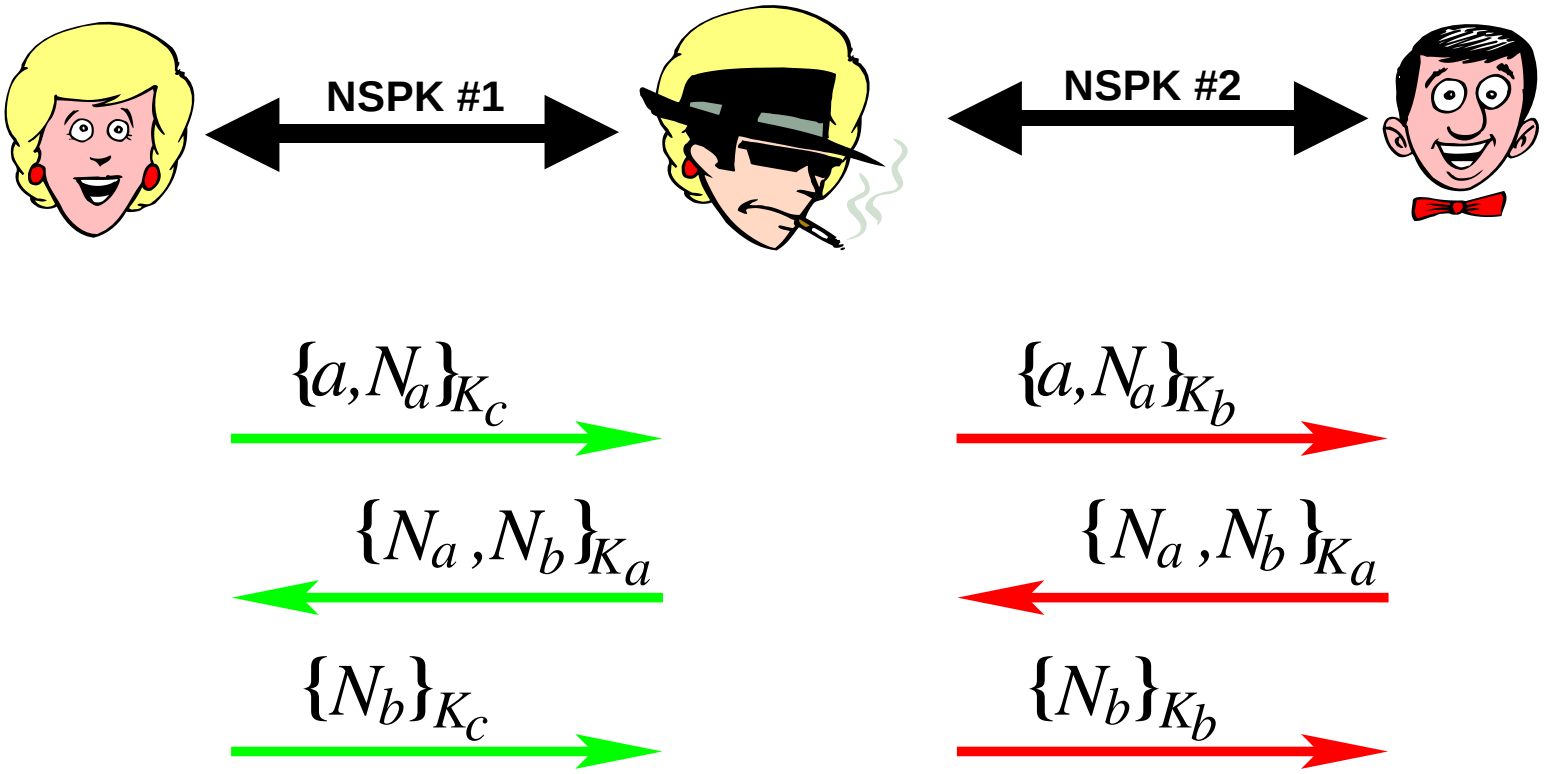
Road map

- Motivation
- Basic notions
- Problems

☞ **Symbolic models**

Recall NSPK

1. $A \to B : \quad \{A, N_A\}_{K_B}$
2. $B \to A : \quad \{N_A, N_B\}_{K_A}$
3. $A \to B : \quad \{N_B\}_{K_B}$



$b(ob)$ **believes he is speaking with** $a(lice)$**!**

What went wrong?

- Problem in step 2.

$$B \rightarrow A : \{N_A, N_B\}_{K_A}$$

Agent B should also give his name: $\{N_A, N_B, \mathbf{B}\}_{K_A}$.

- Is the improved version now correct?

Formal analysis of protocols

- Approach **protocol correctness** as **system correctness**.
- Build a **formal symbolic model** M of protocol.
  - **Formal** = well-defined mathematical semantics.
  - **Symbolic** = abstract away bit-strings to (algebraic) terms.
  - Model as a transition system describing all actions of principals and the attacker.
- Specify property ϕ

  Typically a safety property, e.g., secrecy is an invariant.
- Correctness $M \models \phi$
  - Theorem proving and model checking are main techniques.
  - I will consider each of these in what follows.

Interleaving trace models

- Modeling idea: model possible communication events.

$$
\begin{aligned}
A &\to B : M_1 \\
C &\to D : P_1 \\
B &\to A : M_2 \\
D &\to C : P_2 \\
&\vdots
\end{aligned}
$$

- A **trace** is a sequence of events.
- Trace-based interleaving semantics: **protocol** denotes a trace set.

  Interleavings of (partial) protocol runs and attacker messages.
- Attacker model (Dolev-Yao): the attacker controls the network.

  He can **read**, **intercept**, and **create** messages.

Modeling: protocol as an inductively-defined trace set

$$A \to B : \{A, N_A\}_{K_B}$$
$$B \to A : \{N_A, N_B\}_{K_A}$$
$$A \to B : \{N_B\}_{K_B}$$

Set P formalizes protocol steps.

0. $\langle\rangle \in P$

1. $t, A \to B : \{A, N_A\}_{K_B} \in P$ if $t \in P$ and $fresh_t(N_A)$

2. $t, B \to A : \{N_A, N_B\}_{K_A} \in P$ if $t \in P$, $fresh_t(N_B)$, and $A' \to B : \{A, N_A\}_{K_B} \in t$

3. $t, A \to B : \{N_B\}_{K_B} \in P$ if $t \in P$, $A \to B : \{A, N_A\}_{K_B} \in t$ and $B' \to A : \{N_A, N_B\}_{K_A} \in t$

4. $t, Spy \to B : X \in P$ if $t \in P$ and $X \in has(sees(t))$

Rules 0–3 formalize the protocol steps and rule 4 the attacker model. $sees(t)$ is set of messages in trace t and has[1] is given on next page.

[1] Paulson's formalization uses two inductively defined predicates $synth$ and $analyz$. Account simplified here.

has formalizes analysis powers of DY Intruder

$has(T)$ denotes smallest set of messages inferable from set T

- $t \in T \Rightarrow t \in has(T)$
- $t_1 \in has(T) \wedge t_2 \in has(T) \Rightarrow (t_1, t_2) \in has(T)$
- $(t_1, t_2) \in has(T) \Rightarrow t_i \in has(T)$, for $i \in \{1, 2\}$
- $t_1 \in has(T) \wedge t_2 \in has(T) \Rightarrow \{t_1\}_{t_2} \in has(T)$
- $t_1 \in has(T) \Rightarrow hash(t_1) \in has(T)$
- $\{t_1\}_{t_2} \in has(T) \wedge {t_2}^{-1} \in has(T) \Rightarrow t_1 \in has(T)$

Reflects perfect cryptography assumption:

- Decryption requires right key.
- Hashing collision free and one cannot compute preimage.

Modeling (cont.)

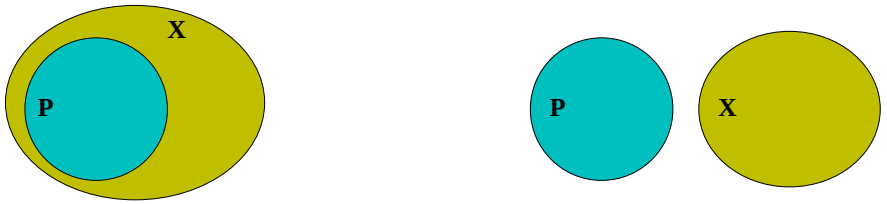
- A **property** also corresponds to **set of traces**.

  Authentication for A: If (1) A used N_A to start a protocol run with B and (2) received N_A back, then B sent N_A back.

$$
\begin{array}{lcll}
AauthenticatesB(t) & \equiv & \textbf{if} & A \rightarrow B : \{A, N_A\}_{K_B} \in t \ \textbf{and} \\
 & & & B' \rightarrow A : \{N_A, N_B\}_{K_A} \in t \\
 & & \textbf{then} & B \rightarrow A : \{N_A, N_B\}_{K_A} \in t \\
SpyattacksA(t) & \equiv & \neg AauthenticatesB(t) &
\end{array}
$$

- Hence the **correctness** of protocols has an exact meaning.

  Every [no] trace of the protocol P has property X.



- Every proposition is either true or false.

  How do we determine which holds?

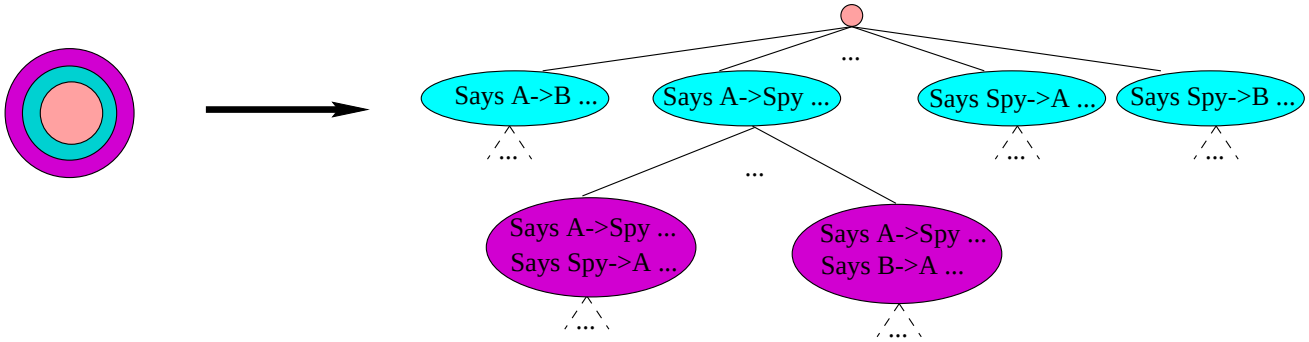
Interactive verification
Inductive approach

- Verification approach due to Larry Paulson, Cambridge
  - Proofs with Isabelle/HOL system (HOL = higher-order logic)
  - Properties established by induction over trace sets
- Many protocols analyzed: TLS, SET, Kerberos IV, ...
  - Proofs scripts constructed by hand (with machine support).
  - Typical protocol requires a few days to construct a proof script
  - Scripts machine-checked, usually in a few minutes
  - Flaws come out in terms of unprovable goals

L C Paulson, The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 1998

Analysis using model-checking

- Inductive definition gives rise to an infinite tree.



- A property correspond to a set of nodes, e.g., $Spyattacks A(t)$.
- **State enumeration** can be used to find attack in the infinite tree, i.e., a node in above set.

  Pure state enumeration is hopelessly inefficient.
- Effective **model-checking** tools exist, e.g., OFMC & Scyther

  Based on **automatic, algorithmic methods**, rather than **interactively constructed proofs** in a formal logic.

Scyther Tool

- Model-checker developed by Cas Cremers.
- Supports automated protocol verification and falsification.
- Performs backward search (from security violation), where an attack is found if initial state is reachable.
- Security properties represented by claim events in the protocol.
- Supports symmetric/asymmetric keys, hash functions, key-tables, multiple protocols in parallel, composed keys, etc.
- Supports different attacker models.
- Extension can generate Isabelle/HOL proofs.
- State-of-art and freely available for download.

Specifying protocols

$$
\begin{aligned}
A \rightarrow B: &\quad \{A, N_A\}_{K_B} \\
B \rightarrow A: &\quad \{N_A, N_B\}_{K_A} \\
A \rightarrow B: &\quad \{N_B\}_{K_B}
\end{aligned}
$$

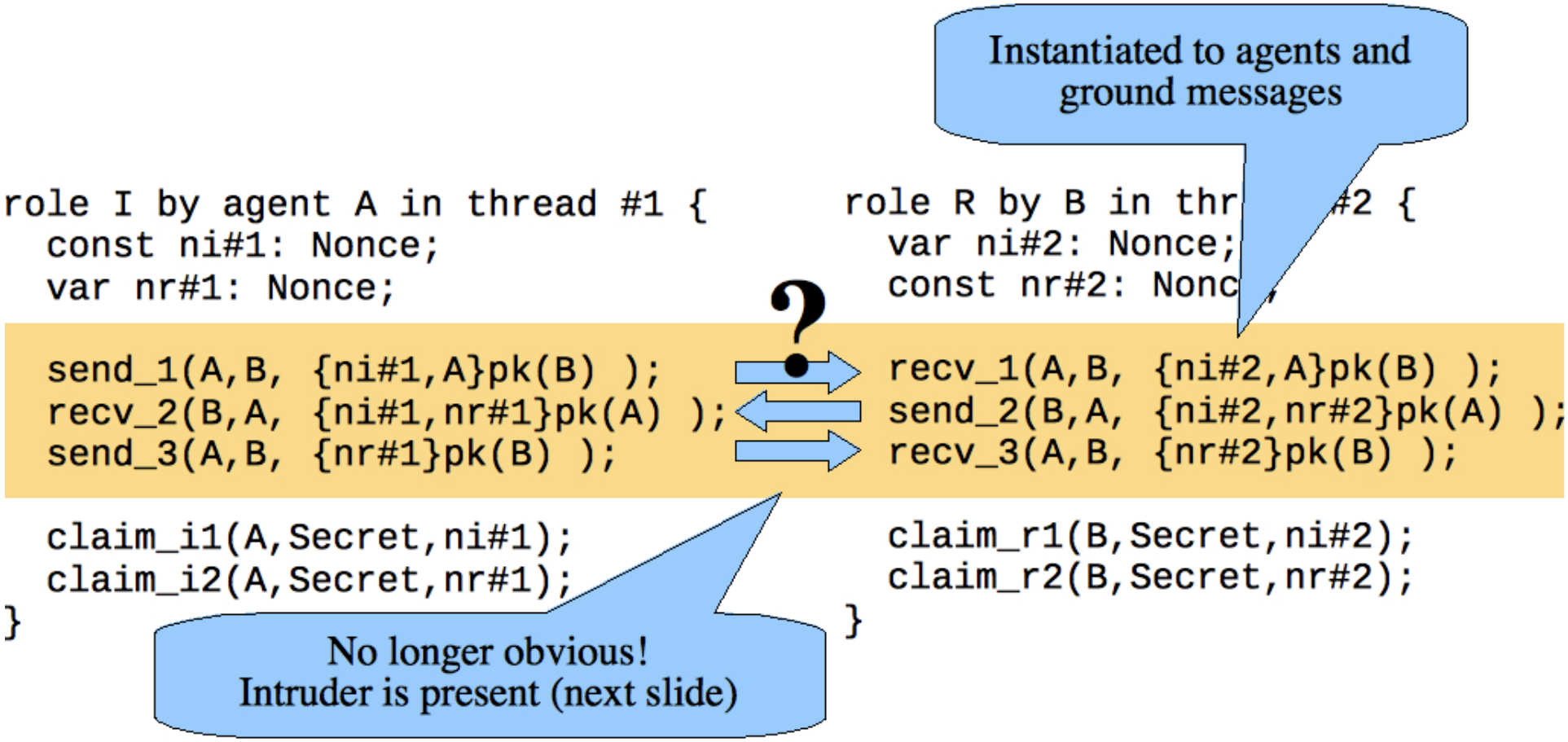
```
const pk: Function;
secret sk: Function;
inversekeys (pk,sk);

protocol ns3(I,R) {

  role I {
    const ni: Nonce;
    var nr: Nonce;

    send_1(I,R, {ni,I}pk(R) );
    recv_2(R,I, {ni,nr}pk(I) );
    send_3(I,R, {nr}pk(R) );

    claim_i1(I,Secret,ni);
    claim_i2(I,Secret,nr);
  }

  role R {
    var ni: Nonce;
    const nr: Nonce;

    recv_1(I,R, {ni,I}pk(R) );
    send_2(R,I, {ni,nr}pk(I) );
    recv_3(I,R, {nr}pk(R) );

    claim_r1(R,Secret,ni);
    claim_r2(R,Secret,nr);
  }

}
```

Declare the existing key infrastructure

Roles and patterns

Insert the security properties that the protocol should satisfy

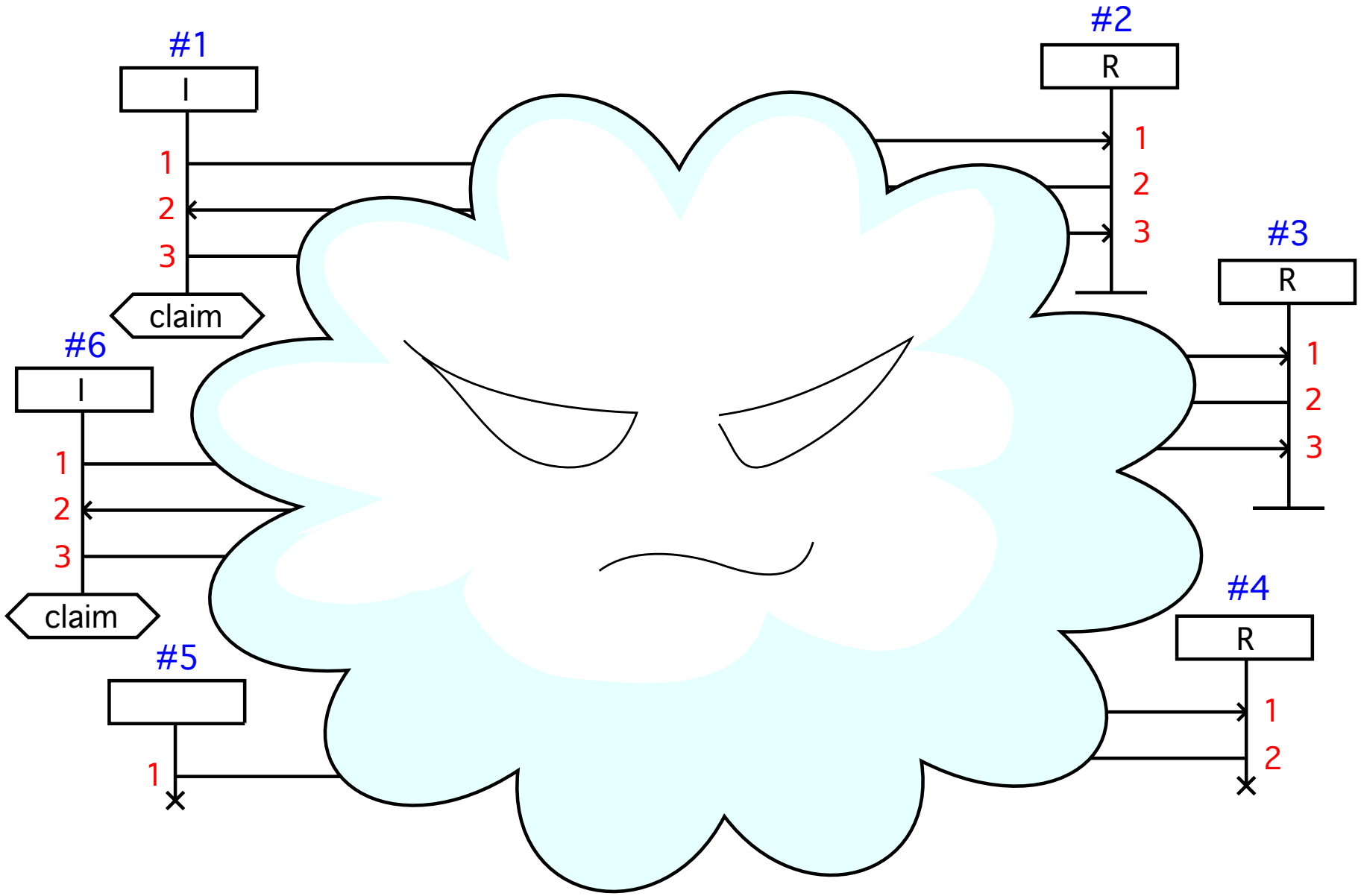
Executing protocols



Each role can be instantiated multiple times.

Each time creates a new thread, with a new identifier, e.g., #1.
Thread variables instantiated during execution.

Executing protocols in the presence of the intruder



Demo of Scyther

Complexity results

- There is no silver bullet!
- **Undecidable** problem in general

  $\Longrightarrow$ No procedure can decide attack/correct for all protocols
- **NP-complete** even when strong constraints are involved.
  - Allow only a bounded number of threads to be created.

  $\Longrightarrow$ State space explosion

Conclusion

- Protocols are tricky to design.

  More needed in practice than secure channels.

- Formal methods provide basis for establishing correctness $P \models \phi$.

- State-of-the-art tools can handle small protocols fairly well.

  Larger protocols, composed of subprotocols, using more complex primitives, are beyond state-of-the-art.

- Correct design $\neq$ correct implementation.
  - Implementation issues must be handled separately.
  - One should not forget model assumptions either.

Reading

- Basin, Mödersheim, Viganò, OFMC: A symbolic model checker for security protocols, *IJIS*, 2004.
- Basin et. al., Improving the Security of Cryptographic Protocols Standards, *IEEE Security & Privacy*, 2015.
- Basin, Cas Cremers, Simon Meier, Provably repairing the ISO/IEC 9798 standard for entity authentication, *Journal of Computer Security*, 2013.
- Basin, Cas Cremers and Catherine Meadows, Model Checking Security Protocols, Handbook of Model Checking, 2015.
- Larry Paulson, The Inductive Approach to Verifying Cryptographic Protocols. *J. Computer Security* 6:85-128, 1998.